National Severe Storms Laboratory

Interface Control Document
for
Radar **I**nterface
and
Data **D**istribution **S**ystem
(RIDDS)

Version 3.10
May 26, 1999

TABLE OF CONTENTS

LIST OF TABLES

1.0 SCOPE

This document defines the interface and data stream which provides base data (Reflectivity, Velocity, and Spectrum Width) from the RIDDS (Radar Interface and Data Distribution System)[1] workstation to external users connected to the RIDDS via a dedicated Ethernet/802.3 Local Area Network (LAN). It will also discuss the difference between the WSR-88D and RIDDS base data stream.

[1] Refer to the RIDDS User's Guide for additional information regarding the RIDDS implementation.

2.0 REFERENCE DOCUMENTS

INTERFACE DOCUMENTS:

1208320I Interface Control Document for Base Data/User
Code Identification 56232
1 November 1992

Source: System Support Branch
Operational Support Facility (South)
3200 Marshall Ave.
Norman, OK 73072

1208321I Interface Control Document for RDA/RPG
Code Identification 56232
1 March 1996

Source: System Support Branch
Operational Support Facility (South)
3200 Marshall Ave.
Norman, OK 73072

TDF 65068 WSR-88D LEVEL II BASE DATA
MAY 1996

Source: National Climatic Data Center (NCDC)
NOAA/NESDIS
151 Patton Avenue
Asheville, NE 28801-5001

3.0 SYSTEM IMPLEMENTATION

The implementation of the RIDDS user interface is divided into two classes: a) local data access, and b) remote data access using the classic "Client-Server" model. All local data access resides on the RIDDS workstation itself and only the system and server processes are officially certified to run on it. No user programs are allowed access to the RIDDS workstation. External user programs will utilize the remote data access method to acquire base data. The RIDDS user interface was specifically designed to:

I. Simulate an independent connection to the WSR-88D base data stream.
II. Provide users Data upon demand.
III. Make available the base data stream to any application running on up to eight workstations connected to the RIDDS ingest host through a multiport (8) Ethernet hub.

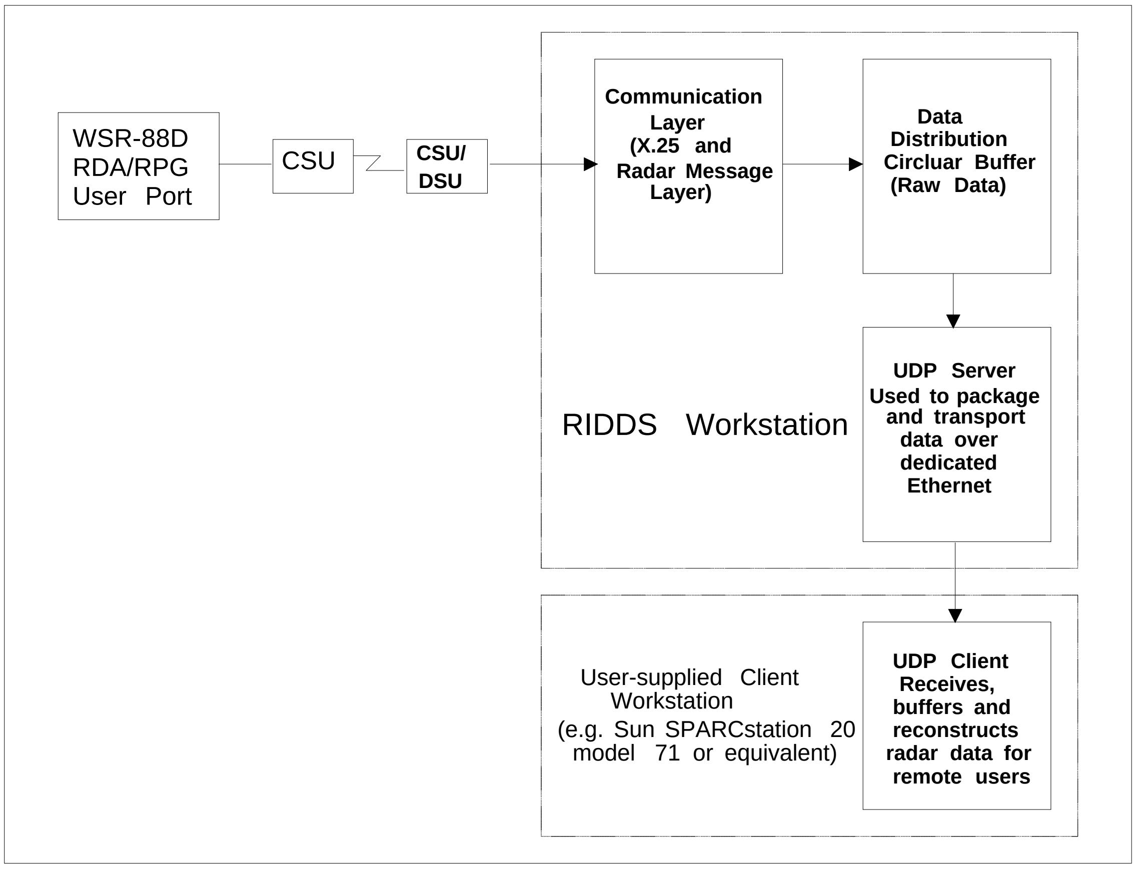
3.1 LOCAL DATA ACCESS

On the RIDDS workstation, a segment of shared memory is allocated to serve as an interim storage location or buffer of the radar data as it is received in real-time. These data are appended to the end of the buffer and a System Pointer is incremented by the number of bytes added. The System Pointer represents the buffer location of the most current radial of radar data. Once the buffer is "full", storage of the data simply rolls off to the beginning of the buffer and the oldest data are overwritten. This buffering model is referred to as a “Circular Buffer”. A user interface was developed consisting of a library of subroutine calls that initiate, maintain, and terminate access to the circular buffer. Refer to Appendix I for more complete details. A local pointer, referred to as the Application Pointer, is used to maintain the current location within the circular buffer that the application is prepared process. As the application requests data from the circular buffer, the System Pointer and the Application Pointer are compared to determine the availability of the data. If the data exists, then they are copied from the circular buffer to a local are a within the application. However, should the data be unavailable or if the data processing is current, the system waits and then attempts again. If the application falls so far behind that the System Pointer overtakes the Application Pointer in the circular buffer, the Application Pointer is reset to the current value of the System Pointer and the application experiences “data shedding”. Data shedding normally should not occur, and its presence might indicate an error in a user program. Since each application maintains its own local pointer, the data request appears to be independent of any other process access in the circular buffer. This benefits all the applications as they can each process the data at their own rates and prevents one application from impacting any other application should it be unable to keep up. The total amount of buffering required depends on the processor speed, data rate, individual application processing efficiency, and the total CPU loading from all applications. The size of the circular buffer is configurable and should be large enough to accommodate one volume scan's worth of data. If the circular buffer is too small, “data shedding” will occur. The current configuration of the ingest host uses a 16MByte shared memory segment configured through the operating systems kernel.

3.2 REMOTE DATA ACCESS

The remote distribution of the WSR-88D base data is achieved through a “User Datagram Protocol” (UDP) Server on the RIDDS workstation. Refer to Appendix II for a detailed description of the RIDDS server and client software. This software should only be started as ROOT or user id (uid) 26. These files must also be setuid to root. The server gains access to the data from the circular buffer, formats the data into UDP packets, and broadcasts these packets over a dedicated Ethernet. Any workstation connected to the dedicated Ethernet and executing the UDP client has the ability to gain access to the data stream. It is important that the network link used for the UDP packets be dedicated for the 88D data stream only. No other network traffic should share this network link. The UDP Client receives the UDP packets, strips the protocol, and reassembles the radar date. The radar data are then stored in an identical fashion as on the RIDDS workstation in a circular buffer. The process to access the data from the circular buffer is implementation can be shown graphically in Figure I.

Figure I. Software Functionality Diagram


WSR-88D RDA/RPG User Port
CSU
CSU/ DSU
Communication Layer (X.25 and Radar Message Layer)
Data Distribution Circluar Buffer (Raw Data)
UDP Server Used to package and transport data over dedicated Ethernet
RIDDS Workstation
User-supplied Client Workstation (e.g. Sun SPARCstation 20 model 71 or equivalent)
UDP Client Receives, buffers and reconstructs radar data for remote users

4.0 DATA FORMAT

The data messages to be transferred between the RDA/RPG to the base data users via the RIDDS workstation are listed in Table II. For a complete list of data message types from the RDA/RPG refer to Appendix III. These messages consist of a message header and the message itself. The message header is used to describe the overall size and type of the message as well as any housekeeping required to send the message over the T1 link. A message header of format specified in Table III is attached to each message transmitted across the link. The maximum number of halfwords to be transferred at one time is 1208 halfwords (2416 byte), including the message header described below. Messages with a length greater than 1208 halfwords are divided into segments of 1208 halfwords or less. The message header of each segment contains both the total number of segments in the entire message and the individual segment number. For messages with lengths of 1208 halfwords or less, the number of message segments is one and the individual segment number is not applicable. Refer to Appendix IV which shows a portion of one packet which includes the message header, and a digital radar data message containing reflectivity only.

In addition to the six data message types from WSR-88D RDA/RPG, there are custom NSSL internal data messages generated by the RIDDS workstation to transmit additional information in the data stream. Table IV shows the structure of the NSSL custom message type.

The RIDDS base data stream is identical to the data stream in the RDA/RPG except for a slight difference in the message header. The message header in the RIDDS base data does not have the "RDA Redundant Channel" filed, which occupies one halfword in byte location 2 in the RDA/RPG base data stream. Instead byte location 2 is used as part of the "Message Type" field in the RIDDS message header, which occupies 2 halfwords from byte location 2 to 3. This is better illustrated in Figure I below.

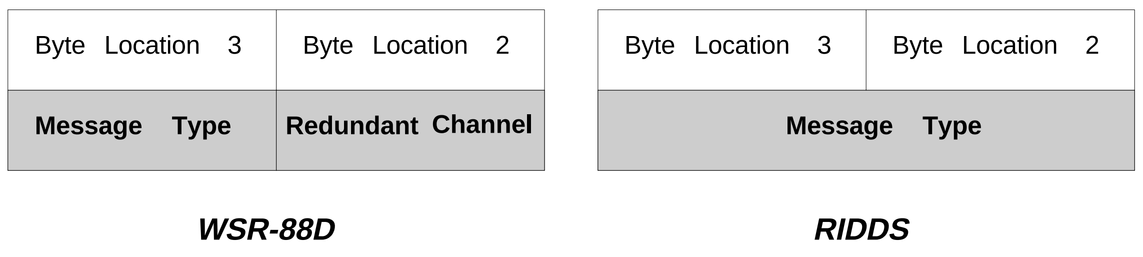


Figure I. Block figure of byte location 2-3 in the WSR-88D and RIDDS message header.

The above difference is only significant in a redundant site where the content of byte locations 2 is not null. However, in a redundant site, the integer value of the message type in byte location 2-3 is offset accordingly depending on the RDA Channel Number. When RDA redundant channel 1 is in effect, the integer value of the message type is offset by an integer value of 256; and when RDA redundant channel 2 is in effect, the message type is offset by an integer value of 512. This is shown in Table I below.

Message Type	***RDA Redundant Channel 1***	***RDA Redundant Channel 2***
1 to 4	256 + 1 to 4	512 + 1 to 4
6 to 14	256 + 6 to 14	512 + 6 to 14
201 to 202	256 + 201 to 202	512 + 201 to 202

Table I. Message Type Offset Value for Redundant Site.

This document was based on the original work by Douglas T. Rhue (1995). We give our sincere appreciation for all his help since then. Please send any suggestions or comments to:

Carl Sinclair, Systems Analyst
National Severe Storms Laboratory
1313 Halley Circle
Norman, OK 73069
ridds@nssl.noaa.gov

TABLE II. DATA MESSAGE TYPE

Type	Message Name	Structure Name	Description
1	Digital Radar Data	wsr88d_drd	NEXRAD Digital Radar Message
2	RDA Status Data	wsr88d_rda	NEXRAD RDA Status Message
20	Console Message	wsr88d_console	NEXRAD Console Message from RPG to Base Data User
21	Console Message	wsr88d_console	NEXRAD Console Message from Base Data User to RPG
22	Loop Back Test	wsr88d_loopback	NEXRAD Loop back Message from RPG to Base Data User
23	Loop Back Test	wsr88d_loopback	NEXRAD Loop back Message from Base Data User to RPG
201	Volume ID	nssl_vol_id	NSSL Message containing current volume number
202	Timestamp	nssl_vol_timestamp	NSSL Message containing current volume number timestamp

TABLE III. MESSAGE HEADER TABLE

The Data messages to be transferred between the RDA/RPG to the Base Data user consist of a message header and the message itself.

The maximum number of halfwords to be transferred at one time is 1208 halfwords (2416 bytes), including the message header described below. Messages with a length greater than 1208 halfwords are divided into segments of 1208 halfwords or less. The message header of each segment contains both the total number of segments in the entire message and the individual segment number. For messages with lengths of 1208 halfwords or less, the number of message segments is one and the individual segment number is not applicable.

Message Header. All WSR88D originated messages contain two parts: 1) the message header and 2) the message itself. The message header is used to describe the overall size and type of the message as well as any housekeeping required to send the message over the T1 link.

Name	**Description**	**Format**	**Units**	**Range**	**Accuracy / Precision**	**Byte Location**
hd_size	Message size in halfwords for this message segment.	Integer	Halfwords	10 to 1208	1	0 - 1
hd_type	Integer Code used to Identify the type of message contained in this segment	Integer		1 to 4, 6 to 14 200 to 250		2 - 3

hd_sequence	Message Sequence Number, 'Serial number' assigned to messages between the RDA and RPG. Since the Base Data user may not see all packets, these values do not appear in order and are ignored by the program.	Integer	ID Number	0 to 32767 then roll over to 1	1	4 - 5
hd_date	Julian Date -2440586.5. Modified Julian Date. Jan. 1 1970 UTC = 1	Integer	Days	1 to 65535	1	6 - 7
hd_time	Number of milliseconds Past Midnight UTC	Integer	Millisecon ds	0 t0 86399999	+/- 2000/1	8 - 11
hd_count	Number of Message segments to send this message	Integer	Segments	1 to 32767	1	12 - 13
hd_segment	Segment number of this message	Integer	Segment ID Number	1 to 32767	1	14 - 15

TABLE IV. NSSL VOLUME DATA MESSAGE

This structure is a NSSL internal message used to pass the current volume number in the data stream. This message is passed to the stream before the first radial of the new volume.

Name	**Description**	**Format**	**Units**	**Range**	**Accuracy / Precision**	**Byte Location**
vol_number	Volume number beginning with the next radial data message	Integer	Volumes	1 to 65535	1	0 - 1

TABLE V. DIGITAL RADAR DATA MESSAGE

This is the proposed enhancement to the standard WSR-88D DRD message for additional parameters. Sufficient room exists in the current format for the addition of 5 additional parameters. Using this format will allow the system to be reverse compatable with LEVELII programs.

The Digital Radar Data Message (DRD) contains the radar reflectivity, velocity and spectrum width fields.

Name	**Description**	**Format**	**Units**	**Range**	**Accuracy / Precision**	**Byte Location**
drd_time	UTC time at which radial data was collected	Integer	Milliseconds	0 to 86399999	± 2000 / 1	0 - 3
drd_date	Current Julian date modified to start January 1, 1970	Integer	Day	1 to 65535	1	4 - 5
drd_range	Unambiguous range, Interval Size, Gate size	Integer (Ref 5)	Km	115 to 511	± 0.1/0.1	6 - 7
drd_az_angle	Azimuth angle at which radial data was collected	Integer (Ref 1, 3)	Degree	0 to 359.956055	± 0.1° / 0.043945°	8 - 9
drd_az_number	Radial Number Within Elevation Cut	Integer (Ref 3)	Count	1 to 400	1	10 - 11
drd_rd_status	Radial Status	Integer (Ref 2)	---	---	---	12 - 13
drd_el_angle	Elevation angle at which radial data was collected.	Integer (Ref 1, 4)	Degree	353° to 70° (-7° to 70°)	± 0.1° / 0.043945°	14 - 15
drd_el_number	Elevation number within volume scan	Integer (Ref 4)	Count	1 to 25	1	16 - 17
drd_sv_range	Range to center of first surveillance gate (BIN)	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	18 - 19

drd_dp_range	Range to center of first doppler gate (BIN)	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	20 - 21
drd_sv_interval	Size of surveillance sample interval (Surveillance gate size)	Integer (Ref 6)	Km	0.25 to 4	± 0.05 / 0.001	22 - 23
drd_dp_interval	Size of doppler sample interval (Doppler gate size)	Integer (Ref 6)	KM	0.25 to 4	± 0.05 / 0.001	24 - 25
drd_sv_count	Number of surveillance gates for current radial	Integer	Count	0 to 460	1	26 - 27
drd_dp_count	Number of Doppler bins for current radial	Integer	Count	0 to 920	1	28 - 29
drd_scan_prof	Sector number within cut	Integer	Count	1 to 3	1	30 - 31
drd_cal_const	Scaling constant used by PSP to calculate reflectivity	32 Bit floating point	DB	-50.0 to 50.0	± 1 / NA	32 - 35
drd_sv_pointer	Pointer to first location of surveillance data in radial	Integer (Ref 8)	Byte	100	1	36 -37
drd_ve_pointer	Pointer to first location of velocity data in radial	Integer (Ref 8)	Byte	100 to 560	1	38 - 39
drd_wt_pointer	Pointer to First location of Spectral width data in radial	Integer (Ref 8)	Byte	100 to 1480	1	40 - 41
drd_ve_resol	Indicated scaling used for Doppler Velocity	Integer	---	---	---	42 - 43
drd_vl_pattern	Identifies Volume Coverage Pattern being used	Integer	---	1 to 767	1	44 - 45
drd_spare_a	Used to V+V Simulator (CPCI 24)	---	---	---	---	46 - 53
drd_a2_ref	Pointer to First location of Surveillance Data in radial, used for Archive II	Integer	Byte	100	1	54 - 55
drd_a2_vel	Pointer to first location of velocity data in radial , used for archive II	Integer	Byte	100 to 560	1	56 - 57

drd_a2_width	Pointer to First location of velocity data in radial used for archive II	Integer	Byte	100 to 1480	1	58 - 59
drd_nyquist	Nyquist Velocity	Integer (Ref 5)	M/S	5.00 to 327.67	± 0.003 / 0.01	60 - 61
drd_atmos	Atmospheric Attenuation Factor	Integer (Ref 5)	dB/Km	-0.02 to -0.002	± 0.004 / 0.001	62 - 63
drd_tover	Threshold Parameter which specifies the minimum difference in echo power between two resolution cells for them to be labeled 'overlayed'	Integer (Ref 5)	dB	0.0 to 20.0	± 0.1 / 0.1	64 - 65
drd_spare_b	Undefined	---	---	---	---	66 - 67
drd_param_count	The number of additional parameters in the message list	Integer	Count	0 to 5	1	68 - 69
drd_p1_range	Range to center of first gate	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	70 - 71
drd_p1_interval	Size of sample or gate interval	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	72 - 73
drd_p1_count	Number of gates	Integer	Count			74 - 75
drd_p2_range	Range to center of first gate	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	76 - 77
drd_p2_interval	Size of sample or gate interval	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	78 - 79
drd_p2_count	Number of gates	Integer	Count			80 - 81
drd_p3_range	Range to center of first gate	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	82 - 83
drd_p3_interval	Size of sample or gate interval	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	84 - 85
drd_p3_count	Number of gates	Integer	Count			86 - 87

drd_p4_range	Range to center of first gate	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	88 - 89
drd_p4_interval	Size of sample or gate interval	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	90 - 91
drd_p4_count	Number of gates	Integer	Count			92 - 93
drd_p5_range	Range to center of first gate	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	94 - 95
drd_p5_interval	Size of sample or gate interval	Integer (Ref 6)	Km	-32.768 to 32.767	± 0.05 / 0.001	96 - 97
drd_p5_count	Number of gates	Integer	Count			98 - 99
drd_rd_data	Weather Radar surveillance data	Integer (Ref 9, 10)	dBZ	-32.0 to 94.5	± 1.0 / 0.5	100 -559
continuation of drd_rd_data	Weather radar velocity data	Integer (Ref 9, 10)	m/sec	-63.5 to 63 -127 to 126	± 1.0 / 0.5 ± 1.0 / 1.0	100 - 1479
continuation of drd_rd_data	Weather radar spectral width data	Integer (Ref 9, 10)	m/sec	-63.5 to 63	± 1.0 / 0.5	100 - 2399
continuation of drd_rd_data	1st Additional Parameter					
continuation of drd_rd_data	2nd Additional Parameter					
continuation of drd_rd_data	3rd Additional Parameter					
continuation of drd_rd_data	4th Additional Parameter					
continuation of drd_rd_data	5th Additional Parameter					

continuation of drd_rd_data	Weather radar spectral width data	Integer (Ref 9, 10)	m/sec	-63.5 to 63	± 1.0 / 0.5	100 - 2399

APPENDIX I. RIDDS SUBROUTINE CALLS

Name:

open_c_buff - create and or attach to a circular buffer located in shared memory

Language:

This routine was developed in the *C* programming language. The calling syntax complies with the *C* calling conventions.

Synopsis:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#include <nssl/c_buff.h>

struct c_buff *open_c_buff(key,size,mode)
int key;
int size;
int mode;
```

Description:

The concept of a circular buffer is a segment of memory that once the end of the allocation is reached the memory is reused by starting at the beginning of the allocation.

In applications using this model, a process controls the data being 'written' to the memory while one or more processes retain control over the amount of data 'read' from the memory. To do this the writing function 'remembers' the next write location though a offset counter referred to as the *highwater marker*. This marker is the offset in bytes from the beginning of the allocated buffer space. The reading function passes to the local process an offset counter referred to as the *lowwater marker*. This marker is the offset in bytes from the beginning of buffer space. Data that has not been read is generally the difference between the high-water marker and the lower marker, when the two markers are equal there is no data to be read.

To setup a circular buffer or to attach to a circular buffer this routine is called. The *key* parameter is used to pass a public access or otherwise a well-known key for services to attach to.

Size is the number of bytes to allocate for the buffer. If this is the writing or otherwise the process that sets up the buffer for the first time, size must contain a positive integer. If this is a reading process this parameter may be zero.

Mode is the read and write permissions to the circular buffer, the values can be found in the description of *shmget().* If the process creates the buffer then the mode value *IPC_CREAT* must be or'd with the mode value.

Diagnostics:

Upon success this routine returns a local process pointer to the shared memory circular buffer, refer to *shmat()* for additional details. Upon failure the routine returns the null pointer.

Possible modes of failure include failure to allocate the shared memory segment or failure to attach to the shared memory segment.

Portability:

This routine is portable to Unix System V workstations configured with share memory and employing the share memory IPC functions.

Name:

close_c_buff - terminate access and detach process from a circular buffer

Language:

This routine was developed in the *C* programming language and adheres to the *C* calling conventions.

Synopsis:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#include <nssl/c_buff.h>

int close_c_buff(circ_buff)
struct c_buff *circ_buff;
```

Description:

This routine closes or terminates the local process access to a circular buffer located in shared memory. Once closed the buffer is detached from the local process, if no other
processes are attached to the buffer an attempt will be made to delete the buffer from the shared memory space.

Circ_buff is a pointer to a previously opened circular buffer using the *open_c_buff()* call.

Diagnostics:

Upon success the routine will return a 0. -1 is returned upon error. If the buffer is not removed from memory, it is not reported as an error.

Portability:

This routine is portable to Unix System V workstations configured with shared memory and employing the shared memory IPC functions.

Name:

read_c_buff - read data from a circular buffer pseudo stream

Language:

This routine was developed using the *C* programming language and adheres to the *C* calling conventions.

Synopsis:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#include <nssl/c_buff.h>

int read_c_buff(circ_buff,buffer,length,lowwater)
struct c_buff *circ_buff;
char *buffer;
int length;
int *lowwater;
```

Description:

This routine is used to retrieve data from a circular buffer.

Circ_buff is a pointer to a shared memory circular buffer created or attached to using the *open_c_buff()* call.

Buffer is a pointer to the data to place the data retrieved from the circular buffer.

Length is the size of the buffer in bytes. When a request for data is given length is the most bytes retrieved. If a message is smaller than length, the entire message will be retrieved. If a message is larger than length the excess will be discarded and lost to this process. Note that messages in excess of 8kbytes are automatically broken into multiple messages.

Lowwater is a pointer to an integer used to maintain the lowwater mark. This value should be initialized by calling *sync_c_buff()*.

Since there is no connection is assumed between the writing and reading processes the circular buffer is exhausted when sufficient bytes are written to exceed the size of the buffer itself. To prevent data that may have been corrupted from entering the reading process, a checksum is calculated and placed in the buffer with the data. When the record is retrieved the reading process will check that data against the checksum to determine if the data is valid.

Diagnostics:

The number of bytes written to the circular buffer is returned upon success. Upon failure the value -1 is returned. If an error occurs the buffer should be synced and tried again.

Portability:

This routine is portable to Unix System V workstations configured with share memory and employing the shared memory IPC functions.

Name:

write_c_buff - write data to a circular buffer pseudo stream

Language:

This routine was developed using the C programming language and adheres to the C calling conventions.

Synopsis:

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>

#include<nssl/c_buff.h>

int write_c_buff(circ_buff,buffer,length)
struct c_buff*circ_buff;
char *buff;
int length;
```

Description:

This routine is used to append data to a circular buffer.

Circ_buff is a pointer to a shared memory circular buffer created to attach to using the *open_c_buff()* call.

Buffer is a pointer to the data to be written to the circular buffer.

Length is the number of bytes pointed to by buffer to be written.

Writing a message to a circular buffer regardless of the content of format forces a record to be written to the buffer. A record consists of a record header and buffer contents.

The record header contains a *record id*, the size of the buffer and a checksum for the buffer.

Since there is no connection assumed between the writing and reading processes the circular buffer is exhausted when sufficient bytes are written to exceed the size of the buffer itself. To prevent data that may have been corrupted from entering the reading process, a checksum is calculated and placed in the buffer with the data. When the record is retrieved the reading process will check the data against the checksum to determine if the data is valid.

Name:

wideband_init_reader - initial routine to attach process to wide band data stream

Language:

This routine was developed in the *'C'* programming language and adheres to the *'C' calling* conventions.

Synopsis:

```
#include <nexrad/riscrpg.h>
#include <nssl/c_buff.h>

struct c_buff *wideband_init_reader(key, lowwater)
int key;
int *lowwater;
```

Description:

This routine will attach and synchronize the process to a shared memory segment used to pass wide band radar data.

Key is the publicly known identifier for the specific shared memory segment desired. Refer to the *RPG* documentation for a list of valid identifiers.

Lowwater is a pointer to the locally maintained low water marker. This marker is used in subsequent calls to *wideband_read()*. Refer to the *c_buff* documentation for additional information on lowwater.

Diagnostics:

This routine will return a pointer to the share memory circular buffer upon success. The NULL pointer is returned upon error. Errors are generally associated with the shared memory segment not be initialized by some other process calling *wideband_init_writer().*

Portability:

Refer to the *c_buff* documentation

Compile Instructions:

The object code for this routine is located in the library file *libnexrad.a*. In order to correctly compile this code the libraries *nssl* and *nexrad* must be included in the compile.

Example:

```
cc myprog.c -I/usr/local/include -L/usr/local/lib -lnssl -lnexrad
```

For additional information refer to the documentation associated with the compiler.

Name:

wideband_init_writer - routine to allocate, initialize and attach process to wide band data stream for writing.

Language:

This routine was developed in the *'C'* programming language and adheres to the *'C' calling* conventions.

Synopsis:

```
#include <nexrad/riscrpg.h>
#include <nssl/c_buff.h>

struct c_buff *wideband_init_writer(key, size)
int key;
int size;
```

Description:

This routine will allocate a segment of shared memory *size* bytes in length for use as a circular buffer. Once allocated this routine will then initialize and attach to the buffer.

Key will be used to identify the buffer to other routines, refer to the *RPG* documentation for information regarding publicly known keys.

Diagnostics:

This routine will return a pointer to the share memory circular buffer upon success. The NULL pointer is returned upon error. Errors are generally associated with the shared memory segment not available or already in use by another process.

Portability:

Refer to the *c_buff* documentation

Compile Instructions:

The object code for this routine is located in the library file *libnexrad.a*. In order to correctly compile this code the libraries *nssl* and *nexrad* must be included in the compile.

Example:

```
cc myprog.c -I/usr/local/include -L/usr/local/lib -lnssl -lnexrad
```

For additional information refer to the documentation associated with the compiler.

Name:

wideband_read - read a 'message' from a wideband circular buffer

Language:

This routine was developed in the *'C'* programming language and adheres to the *'C'* calling conventions.

Synopsis:

```
#include <nexrad/riscrpg.h>
#include <nssl/c_buff.h>

int wideband_read(circ_buff, msg_buff, msg_bufflen, lowwater)
struct c_buff *circ_buff;
char msg_buff;
int msg_bufflen;
int *lowwater;
```

Description:

This routine reads the wideband circular buffer pointed to by *circ_buff* and attempts to retrieve messages written to the circular buffer by *wideband_write()*.

Circ_buff is a pointer to the desired circular buffer. Generally *circ_buff* will be a value returned by *wideband_init_reader()*.

Name:

wideband_write - write a 'message' to a wide band circular buffer

Language:

This routine was developed in the *'C'* programming language and adheres to the *'C'* calling conventions.

Synopsis:

```
#include <nexrad/riscrpg.h>
#include <nssl/c_buff.h>

int wideband_write(circ_buff, msg_buff, msg_len)
struct c_buff *circ_buff;
char *msg_buff;
int msg_len;
```

Description:

This routine writes a message to the circular buffer pointed to by *circ_buff.*

Circ_buff is a pointer to the desired circular buffer. Generally *circ_buff* will be a value returned by *wideband_init_writer()*.

Msg_buff is a pointer to the message to be written to the wide band circular buffer.

Msg_len is the length of the *msg_buff* in bytes. *Msg_bufflen* should not exceed 2500 bytes in length.

Diagnostics:

The number of bytes written is returned upon success, -1 is returned upon error.

Portability:

Refer to the *c_buff* documentation.

Compile Instructions:

Refer to *wideband_init_writter()*.

APPENDIX II. RIDDS CLIENT SERVER SOFTWARE

Name:
widebd_server - distribute WSR88D data over Ethernet using UDP broadcast

Language:
This routine was developed in the '*C'* programming language. This routine is a Unix command and adheres to the Unix command line conventions.

Synopsis:
```
widebd_server <ip address> <port number>
```

Description:
This program provides the mechanical interface between the WSR88D circular buffer and the network interface used for data broadcast. The basic process is to retrieve a message from the circular buffer, break it up into UDP packets and send the packets over the network.

Ip address is the broadcast address to send the data to. While this utility is intended to broadcast the data to the network, it may be used to send data point to point. When using the point to point mode nothing is gained over the broadcast mode.

The default *ip address* used by the program is 129.15.47.255.

Port number is the port number associated with the broadcast stream. If a specific port is desired at the *ip address* must be supplied as well.

The default *port number* is 3280.

Diagnostics:
This utility will return a 1 status upon error, and a 0 status upon completion. Errors will be reported as error messages on the standard error stream (stderr).

Name:

widebd_client - receive WSR88D data over Ethernet using UDP broadcast

Language:

This routine was developed in the '*C'* programming language. This routine is a Unix command and adheres to the Unix command line conventions.

Synopsis:

```
widebd_client <ip address> <port number>
```

Description:

This program provides the mechanical interface between the network interface used to broadcast WSR88D data and the local circular buffer. The basic process is to receive packets from the network, reconstruct the packets into messages and place the messages into the circular buffer. Local applications needing the data then process the circular buffer for the messages.

Ip address is the broadcast address to receive data from. While this utility is intended to receive broadcast data from the network, it may be used to receive point to point data. When using the point to point mode nothing is gained over the broadcast mode.

The default *ip address* used by the program is 129.15.47.255.

Port number is the port number associated with the broadcast stream. If a specific port is desired at the *ip address* must be supplied as well.

The default *port number* is 3280.

Diagnostics:

This utility will return a 1 status upon error, and a 0 status upon completion. Errors will be reported as error messages on the standard error stream (stderr).

APPENDIX III. DATA MESSAGE TYPES

Message Type	Description
1	Digital Radar Data
2	RDA Status Data
3	Performance/Maintenance Data
4	Console Message - RDA to RPG
5	Maintenance Log Data
6	RDA Control Commands
7	Volume Coverage Pattern
8	Clutter Censor Zones
9	Request for Data
10	Console Message - RPG to RDA
11	Loop Back Test - RDA to RPG
12	Loop Back Test - RPG to RDA
13	Clutter Filter Bypass Map - RDA to RPG
14	Edited Clutter Filter Bypass Map
15	Clutter Filter Notch Width Map
20	Console Message - RDA (or RPG) to User
21	Console Message - User to DEA (or RPG)
22	Loop Back Test - RDA (or RPG) to User
23	Loop Back Test - USER to RDA (or RPG)

Referenced from page 5-2 in the Interface Control Document for RDA/RPG (Document No. 1208321I; Code Identification 56232; 1 March 1996)